26장 ES6 함수의 추가 기능

26.1 함수의 구분

- ES6 이전까지

  사용 목적에 따라 명확히 구분되지 않는다. 즉, ES6 이전의 모든 함수는 일반함수로서 호출할 수 있는 것은 물론 생성자 함수로서 호출할 수 있다.

  ⇒ 즉, ES6 이전의 모든 함수는 callable 이면서 constructor

- ES6 이후부터

  사용 목적에 따라 세 가지 종류로 명확히 구분

ES6 함수의 구분	constructor	prototype	super	arguments
일반 함수	O	O	X	O
메서드	X	X	O	O
화살표 함수	X	X	X	X

26.2 메서드

- ES6 사양에서 메서드 : 메서드 축약 표현으로 정의된 함수
  - 인스턴스를 생성할 수 없는 non-constructor
    - 따라서 ES6 메서드는 생성자 함수로서 호출할 수 없다.

```
const obj = {
    x: 1,
  // foo는 메서드다.
  foo () { return this.x; }
  // bar에 바인딩된 함수는 메서드가 아닌 일반 함수다.
  bar: function () { return this.x; }
}
```

```
new obj.foo(); // TypeError
new obj.bar(); // bar {}
```

- ES6 메서드는 인스턴스를 생성할 수 없으므로 prototype 프로퍼티가 없고 프로토타입도 생성하지 않는다.
- ES6 메서드는 자신을 바인딩한 객체를 가리키는 내부 슬롯 [[HomeObject]]를 갖는다.

26.3 화살표 함수

- 화살표 함수(arrow function)
  - function 키워드 대신 화살표(=>)를 사용하여

    기존의 함수 정의 방식보다 간략하게 함수를 정의

화살표 함수 정의

```
const arrow = (x, y) => x + y;
```

- 화살표 함수는 함수 선언문으로 정의할 수 없고 함수 표현식으로 정의해야 한다.

```
const arrow = (x, y) => { ... };

const arrow = x => { ... };
```

- 매개변수가 여러 개인 경우 소괄호 () 안에 매개변수를 선언한다.
- 한 개인 경우 소괄호 ()를 생략할 수 있다.
- 매개변수가 없는 경우 소괄호 ()를 생략할 수 없다.

```
const power = x => x ** 2;
power(2); // 4

const arrow = () => const x = 1; // ❌ SyntaxError
```

- 함수 몸체가 하나의 문으로 구성된다면 {}를 생략할 수 있다.
- 함수 몸체 내부의 문이 표현식이 아닌 문이라면 에러가 발생한다.

```
const create = (id, content) => ({ id, content });
```

- 객체 리터럴을 반환하는 경우 객체 리터럴을 소괄호 ()로 감싸 주어야 한다.

```
const person = (name => ({
  sayHi() { return name; }
}))('Cho');

console.log(person.sayHi()); // Cho
```

- 화살표 함수도 즉시 실행 함수로 사용할 수 있다.

화살표 함수와 일반 함수의 차이

- 화살표 함수는 인스턴스를 생성할 수 없는 non-constructor다.
- 중복된 매개변수 이름을 선언할 수 없다.
- 화살표 함수는 함수 자체의 this, arguments, super, new.target 바인딩을 갖지 않는다.

> 💡 this

화살표 함수는 함수 자체의 this 바인딩을 갖지 않는다.

따라서 화살표 함수 내부에서 this를 참조하면 상위스코프의 this를 그대로 참조하는데. 이를 lexical this라 한다.

이는 마치 렉시컬 스코프와 같이 화살표 함수의 this가 함수가 정의된 위치에 의해 결정된다는 것을 의미한다.

```
const foo = () => console.log(this);
foo(); // window
```

- 화살표 함수가 전역 함수라면 화살표 함수의 this는 전역 객체를 가리킨다.
- 화살표 함수는 함수 자체의 this 바인딩을 갖지 않기 때문에 화살표 함수 내부의 this를 교체할 수 없다.
- 메서드를 화살표 함수로 정의하는 것은 피해야 한다. 메서드를 정의할 때는 ES6 메서드 축약 표현으로 정의한 ES6 메서드를 사용하는 것이 좋다.
- 프로토타입 객체의 프로퍼티에 화살표 함수를 사용하지 않는다.

super

화살표 함수는 함수 자체의 super 바인딩을 갖지 않는다. 따라서 화살표 함수 내부에서 super를 참조하면 this와 마찬가지로 상위 스코프의 super를 참조한다.

arguments

화살표 함수는 함수 자체의 arguments 바인딩을 갖지 않는다. 따라서 화살표 함수 내부에서 arguments를 참조하면 this와 마찬가지로 상위 스코프의 arguments를 참조한다. 따라서 화살표 함수로 가변 인자 함수를 구현해야 할 때는 반드시 Rest 파라미터를 사용해야 한다.

26.4 Rest 파라미터

- Rest 파라미터(나머지 매개변수)

매개변수 이름 앞에 세개의 점 ...을 붙여서 정의한 매개변수를 의미한다. Rest 파라미터는 함수에 전달된 인수들의 목록을 배열로 전달받는다.

```
function foo(...rest) {
  console.log(rest); // [1, 2, 3, 4, 5]
}

foo(1, 2, 3, 4, 5);
```

- 일반 매개변수와 Rest 파라미터는 함께 사용할 수 있다.
- Rest 파라미터는 반드시 마지막 파라미터이어야 한다.

- Rest 파라미터는 단 하나만 선언할 수 있다.
- Rest 파라미터는 함수 정의 시 선언한 매개변수 개수를 나타내는 함수 객체의 length 프로퍼티에 영향을 주지 않는다.
- ES6에서는 rest 파라미터를 사용하여 가변 인자 함수의 인수 목록을 배열로 직접 전달받을 수 있다.

26.5 매개변수 기본값

```
function sum (x = 0, y = 0) {
  return x + y;
}

console.log(sum(1, 2)); // 3
console.log(sum(1)); // 1

console.log(sum.length); // 0
```

- Rest 파라미터에는 기본값을 지정할 수 없다.